Modern Statistics for Modern Biology: Clustering

Data Science Seminar

Greg Botwin

Cedars-Sinai

2019-MAY-08

Clustering Overview

- Takes data (continuous or quasi-continuous) and adds to them a new categorical group variable
- Can assist in simplifying decision making, but comes at a cost of ignoring intermediate states
- caveat: clustering algorithms are designed to find clusters, so they will find clusters, even where there are none
    - cluster validation is an essential component of our process, especially if there is no prior domain knowledge that supports the existence of clusters.

How do we measure similarity

- Decide what we mean by similar (e.g. similar by height or similar by habitat)
- Chose how to combine features into a single number
    - Do they have to be in the same scale?
- Mathematical notation distance between two points $A = (a_1, \ldots, a_p)$ and $B = (b_1, \ldots, b_p)$ in p dimensionsal space.

Which of the two cluster centers is the red point closest to?

Test Data

```
test <- data.frame("x" = c(0,0,0,1,1,1),
                   "y" = c(1,0,1,1,0,1),
                   "z" = c(1,1,1,0,1,1))

test
```

```
#   x y z
# 1 0 1 1
# 2 0 0 1
# 3 0 1 1
# 4 1 1 0
# 5 1 0 1
# 6 1 1 1
```

- 6 points A, in 3 dimensions p

Parameters to dist() Function

```
# to use less space than n^2 positions,
# by default returns only lower triangle
dist(test)
```

```
#          1        2        3        4        5
# 2 1.000000
# 3 0.000000 1.000000
# 4 1.414214 1.732051 1.414214
# 5 1.414214 1.000000 1.414214 1.414214
# 6 1.000000 1.414214 1.000000 1.000000 1.000000
```

```
# by default "euclidian" method
dist(test)

# also accepts, euclidean", "maximum", "manhattan", "canberra", "bina

dist(test, method = "binary")
```

Full Similarity Matrix

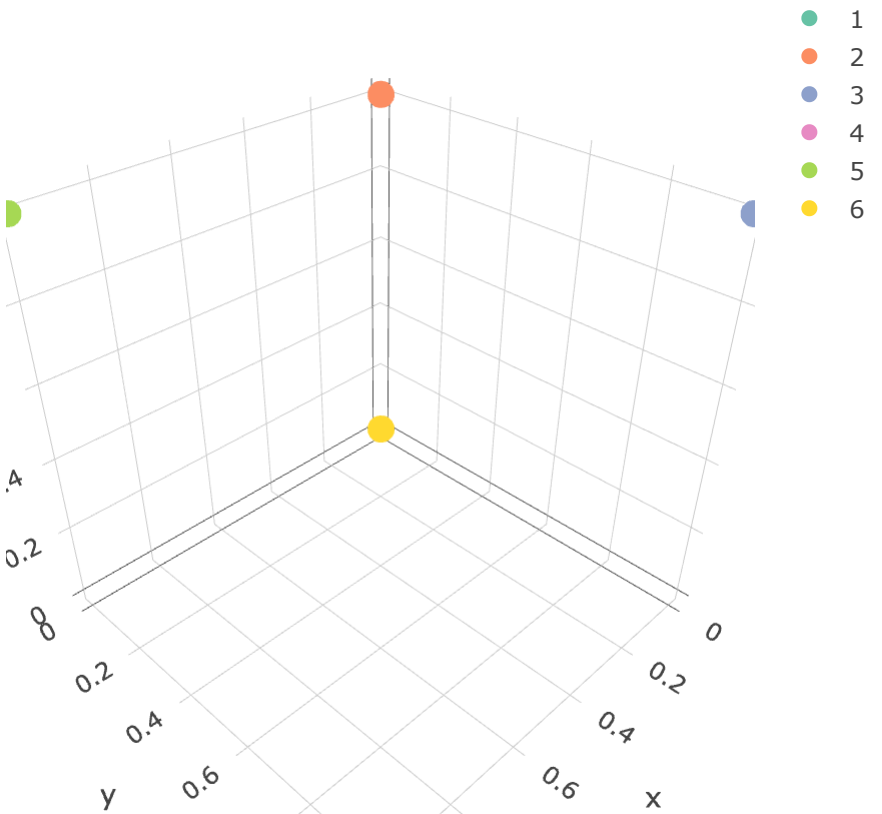
```
dist(test, diag = TRUE, upper = TRUE)
```

```
##          1        2        3        4        5        6
## 1 0.000000 1.000000 0.000000 1.414214 1.414214 1.000000
## 2 1.000000 0.000000 1.000000 1.732051 1.000000 1.414214
## 3 0.000000 1.000000 0.000000 1.414214 1.414214 1.000000
## 4 1.414214 1.732051 1.414214 0.000000 1.414214 1.000000
## 5 1.414214 1.000000 1.414214 1.414214 0.000000 1.000000
## 6 1.000000 1.414214 1.000000 1.000000 1.000000 0.000000
```

- Note 0 perfectly similar

Computations related to distance in R Types

```
plot_ly(test, x = ~x, y = ~y, z = ~z,
        color = rownames(test))
```



Euclidean Similarity

$$d(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_p - b_p)^2}$$

- is the square root of the sum of squares of the differences in all p coordinate directions, also called L2

```
dist(test)
```

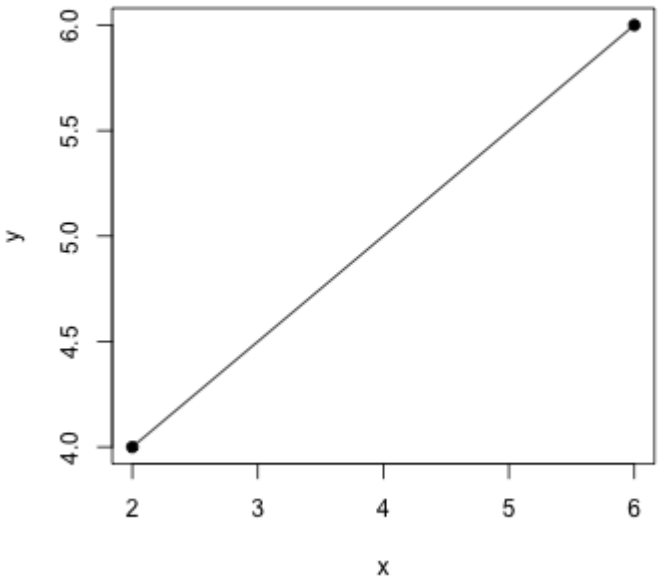
```
##          1        2        3        4        5
## 2 1.000000
## 3 0.000000 1.000000
## 4 1.414214 1.732051 1.414214
## 5 1.414214 1.000000 1.414214 1.414214
## 6 1.000000 1.414214 1.000000 1.000000 1.000000
```

Euclidean Distance (simple example)

```
#example
example_1 <- data.frame(x=c(2,6), y=c(4,6))
example_1
```

```
##   x y
## 1 2 4
## 2 6 6
```

Euclidean Distance (simple example)




```
dist(example_1)
```

```
##          1
## 2 4.472136
```

Maximum Distance

$$d_\infty(A, B) = \max_i |a_i - b_i|.$$

- The maximum of the absolute differences between coordinates is also called the L_∞ distance:

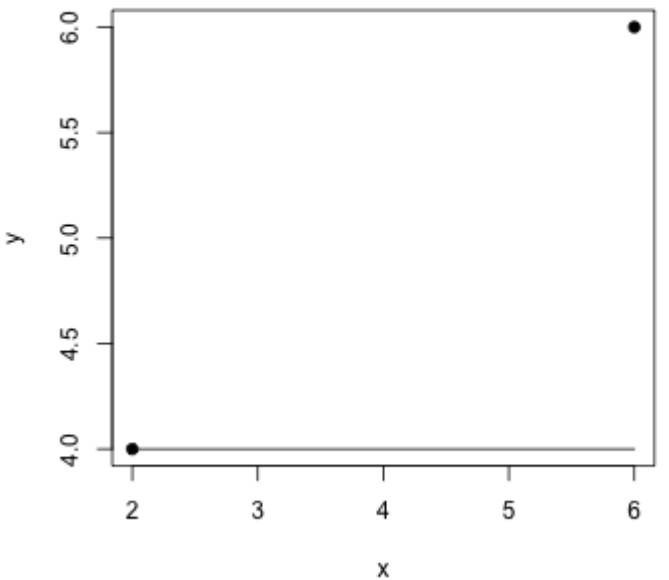
```
dist(test, method = "maximum")
```

```
##   1 2 3 4 5
## 2 1
## 3 0 1
## 4 1 1 1
## 5 1 1 1 1
## 6 1 1 1 1 1
```

Maximum Distance (simple example)

```
dist(example_1, method = "maximum")
```

```
##   1
## 2 4
```



Manhattan Distance

$$d(A, B) = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_p - b_p|.$$

- The Manhattan, City Block, Taxicab or L1 distance takes the sum of the absolute differences in all coordinates

```
test[1:3,]
```

```
##   x y z
## 1 0 1 1
## 2 0 0 1
## 3 0 1 1
```

```
dist(test, method = "manhattan")
```

```
##   1 2 3 4 5
## 2 1
## 3 0 1
## 4 2 3 2
## 5 2 1 2 2
## 6 1 2 1 1 1
```

Weighted Euclidean Distance

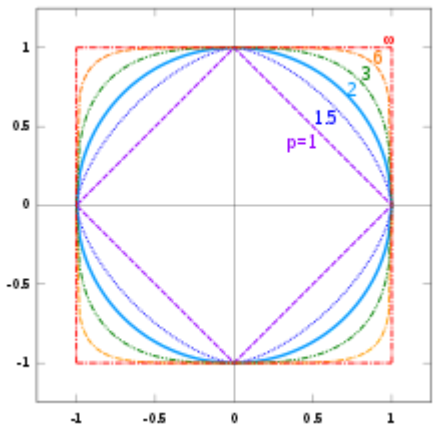
- Generalization of the ordinary Euclidean distance, by giving different directions in feature space different weights
- Mahalanobis distance is a Euclidean distance that takes into account the fact that different features may have a different dynamic range, and that some features may be positively or negatively correlated with each other. The weights in this case are derived from the covariance matrix of the features.

Minkowski Distance

$$d(A,B) = ((a_1 - b_1)^m + (a_2 - b_2)^m + \ldots + (a_p - b_p)^m)^{\frac{1}{m}}.$$

- Generalization allowing the exponent to be m
- If m = 1, same as manhattan
- If m = 2, same as euclidian

```
dist(test, method = "minkowski", p = 1.5)
```



Hamming Distance

- This distance is the simplest way to compare character sequences.
- It simply counts the number of differences between two character strings
- "AAGGCC**TT**" vs "AAGGCC**AA**" = 2

```
library(Biostrings)
stringDist(c("AAGGCCTT", "AAGGCCAA"), method = "hamming")
```

```
##   1
## 2 2
```

Binary Distance

- non-zero elements treated as ‘on’ and the zero elements as ‘off’
- computes the proportion of features having only one bit on amongst those features that have at least one bit on

```
x <- c(0, 0, 1, 1, 1, 1)
y <- c(1, 0, 1, 1, 0, 1)
# answer = 2/5
dist(rbind(x, y), method = "binary")
```

```
##     x
## y 0.4
```

```
x <- c(1, 1, 1, 1, 1, 1)
y <- c(0, 0, 0, 0, 0, 0)
# answer = 6/6
dist(rbind(x, y), method = "binary")
```

```
##   x
## y 1
```

Jaccard Index

$$J(S,T) = \frac{f_{11}}{f_{01} + f_{10} + f_{11}},$$

- Occurrence of traits or features can be translated into presence and absence and encoded as 1's and 0's
- Use if co-existence is more important than co-absence (e.g. mutation patterns)
- f_{11} = the number of times a feature co-occurs in S and T
- f_{10} and f_{01} the number of times a feature occurs in S but not in T (and vice versa)
- f_{00} he number of times a feature is co-absent
- dissimilarity is simply 1 - Jaccard Index

```
s <- c(0,0,1,1,1,1)
t <- c(1,0,1,1,0,1)
# similarity
3 / (1 + 1 + 3)
```

```
## [1] 0.6
```

Jaccard Distance Example

```
mut <- read.csv("data/HIVmutations.csv")
dim(mut)
```

```
## [1]  5 57
```

```
mut[1:3, 10:16]
```

```
##   p32I p33F p34Q p35G p43T p46I p46L
## 1    0    1    0    0    0    0    0
## 2    0    1    0    0    0    1    0
## 3    0    1    0    0    0    0    0
```

Jaccard Distance Example

```
library(vegan)
vegdist(mut, meto = "jaccard")
```

```
##           1         2         3         4
## 2 0.6666667
## 3 0.6000000 0.8000000
## 4 0.8181818 0.6363636 0.7333333
## 5 1.0000000 0.6666667 0.8000000 0.8181818
```

```
as.dist(sqrt(2 * (1 - cor(t(mut)))))
```

```
##          1        2        3        4
## 2 1.186342
## 3 1.104026 1.302931
## 4 1.318368 1.133893 1.298780
## 5 1.452966 1.186342 1.302931 1.318368
```

Non-numeric Feature Space

- General dissimilarity coefficient of Gower you are able to handle other variable types as well (e.g. nominal, ordinal, (a)symmetric binary) even when different types occur in the same data set.
- The dissimilarity between two rows is the weighted mean of the contributions of each variable

```
library(cluster)
library(ggplot2)
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

Gower Dissimilarity

```
daisy(head(diamonds), metric = "gower")
```

```
## Dissimilarities :
##           1         2         3         4         5
## 2 0.2329529
## 3 0.4240170 0.3126328
## 4 0.4974255 0.5537117 0.5950373
## 5 0.6600680 0.7663542 0.7243464 0.2626424
## 6 0.4344866 0.4984852 0.5151703 0.3343932 0.4578689
##
## Metric :  mixed ;  Types = I, O, O, O, I, I, I, I, I, I
## Number of objects : 6
```

When to scale

- If variables are not scaled
  - variable with largest range has most weight
  - distance depends on scale
- Scaling gives every variable equal weight
- Scale if,
  - variables measure different units (kg, meter, sec,…)
  - you explicitly want to have equal weight for each variable

Nonparametric mixture detection

- Work well in high-dimensional settings, where we cannot easily use probability densities, the EM algorithm and parametric mixture
- Besides the distance measure, the main choice to be made is the number of clusters k
- The centers of the groups are sometimes called medoids, thus the name PAM (partitioning around medoids)

Steps in PAM

1. Starts from a matrix of p features measured on a set of n observations.
2. Randomly pick k distinct cluster centers out of the n observations (“seeds”).
3. Assign each of the remaining observation to the group to whose center it is the closest.
4. For each group, choose a new center from the observations in the group, such that the sum of the distances of group members to the center is minimal; this is called the medoid.
5. Repeat Steps 3 and 4 until the groups stabilize.

PAM vs k-means

- k-means replaces the medoids by the arithmetic means (centers of gravity) of the clusters
- In PAM, the centers are observations, this is not, in general, the case with k-means.
- Both work well when the clusters are of comparable size and convex (blob-shaped)
- Poor performance if the true clusters are very different in size, the larger ones will tend to be broken up; or they have pronounced non-spherical or non-elliptic shapes.

Tight clusters with resampling

- Repeating a clustering procedure multiple times on the same data, but with different starting points creates strong forms
- Repeated subsampling of the dataset and applying a clustering method will result in groups of observations that are “almost always” grouped together; these are called tight clusters

Question 5.4

```
library(clusterExperiment)
library(scRNAseq)
library(SummarizedExperiment)
data("fluidigm", package = "scRNAseq")

assay(fluidigm)[1:5,1:5]
```

```
##           SRR1275356 SRR1274090 SRR1275251 SRR1275287 SRR1275364
## A1BG               0          0          0          0          0
## A1BG-AS1           0          0          0          0          0
## A1CF               0          0          0          0          0
## A2M                0          0          0         31          0
## A2M-AS1            0          0          0          0          0
```

```
NROW(fluidigm) # number of genes
```

```
## [1] 26255
```

Sample Metadata

```
SummarizedExperiment::colData(fluidigm)[,1:5]
```

```
## DataFrame with 130 rows and 5 columns
##                NREADS  NALIGNED    RALIGN TOTAL_DUP    PRIMER
##            <numeric> <numeric> <numeric> <numeric> <numeric>
## SRR1275356  10554900   7555880   71.5862   58.4931 0.0217638
## SRR1274090    196162    182494   93.0323   14.5122 0.0366826
## SRR1275251   8524470   5858130   68.7213   65.0428 0.0351827
## SRR1275287   7229920   5891540   81.4884   49.7609 0.0208685
## SRR1275364   5403640   4482910   82.9609   66.5788 0.0298284
## ...              ...       ...       ...       ...       ...
## SRR1275259   5949930   4181040   70.2705   52.5975 0.0205253
## SRR1275253  10319900   7458710   72.2747   54.9637 0.0205342
## SRR1275285   5300270   4276650   80.6873   41.6394 0.0227383
## SRR1275366   7701320   6373600     82.76   68.9431 0.0266275
## SRR1275261  13425000   9554960   71.1727   62.0001 0.0200522
```

```
NCOL(fluidigm) #number of samples
```

```
## [1] 130
```

Filtering and Normalization

```
# limit the analysis to the samples corresponding to high sequencing
se <- fluidigm[,colData(fluidigm)[,"Coverage_Type"]=="High"]

# retain only those genes with at least 10 reads in at least 10 cells
wh_zero <- which(rowSums(assay(se))==0)
pass_filter <- apply(assay(se), 1, function(x) length(x[x >= 10]) >=
se <- se[pass_filter,]
dim(se)
```

```
## [1] 7069   65
```

```
# quantile normalization
fq <- round(limma::normalizeQuantiles(assay(se)))
assays(se) <- list(normalized_counts=fq)
```

```
#update names
wh<-which(colnames(colData(se)) %in% c("Cluster1","Cluster2"))
colnames(colData(se))[wh]<-c("Published1","Published2")
```

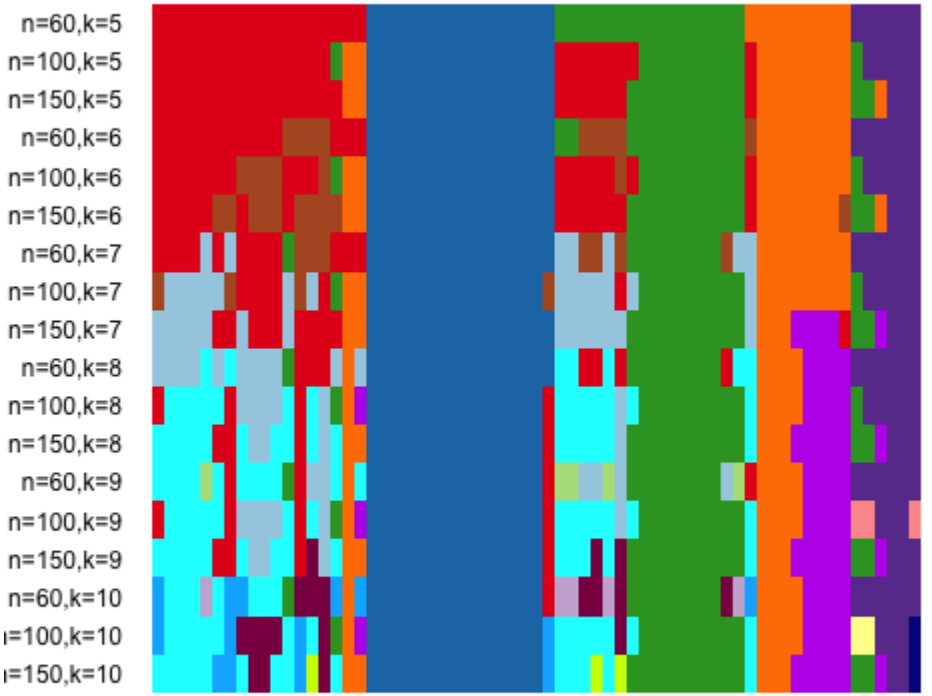
Perform Clustering

- Set the choice of genes to include at either the 60, 100 or 150 most variable genes.

-Plot the clustering results for k varying between 4 and 9

```
ce = clusterMany(se, clusterFunction = "pam", ks = 5:10, run = TRUE,
  isCount = TRUE, reduceMethod = "var", nFilterDims = c(60, 100, 150)
clusterLabels(ce) = sub("FilterDims", "", clusterLabels(ce))
plotClusters(ce, whichClusters = "workflow", axisLine = -1)
```

Cluster Plot (cell = column)



Flow cytometry and mass cytometry

```
library("flowCore")
library("flowViz")

fcsB = read.FCS("data/Bendall_2011.fcs")

slotNames(fcsB)
```

```
## [1] "exprs"       "parameters"  "description"
```

```
# How many variables were measured?
head(colnames(fcsB))
```

```
## [1] "Time"              "Cell_length"       "Ir(190.960)-Dual"
## [4] "Ir(192.962)-Dual" "Rh(102.905)-Dual" "In(114.903)-Dual"
```

```
# How many Cells Were Measures
dim(exprs(fcsB))
```

```
## [1] 91392    41
```

Data preprocessing

```
#match isotype name to marker name
markersB = readr::read_csv("data/Bendall_2011_markers.csv")
head(markersB)
```

```
## # A tibble: 6 x 2
##   isotope          marker
##   <chr>            <chr>
## 1 Nd(144.912)-Dual CD4
## 2 Nd(145.913)-Dual CD8
## 3 Sm(146.914)-Dual CD20
## 4 Gd(157.924)-Dual CD33
## 5 Er(169.935)-Dual CD56
## 6 Ir(190.960)-Dual DNA191
```

```
mt = match(markersB$isotope, colnames(fcsB))
stopifnot(!any(is.na(mt)))
colnames(fcsB)[mt] = markersB$marker
```

Clustering on One Marker

```
flowPlot(fcsB, plotParameters = colnames(fcsB)[2:3], logy = TRUE)
```

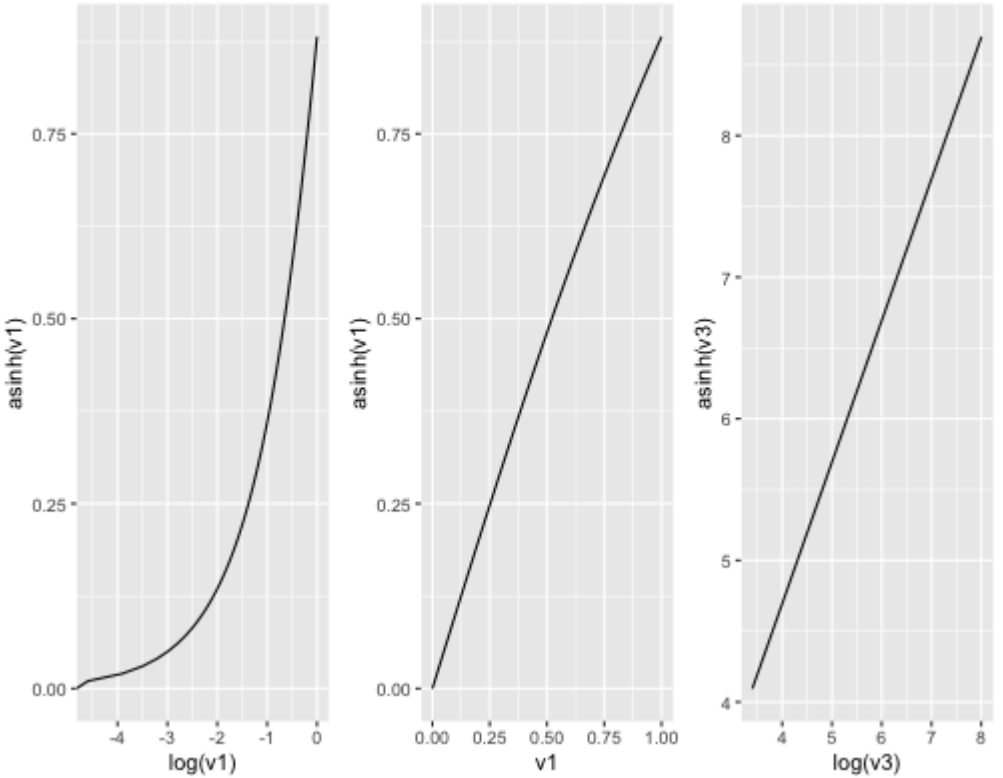
Data Transformation

$$\operatorname{asinh}(x) = \log\left(x + \sqrt{x^2 + 1}\right).$$

- for large values of x , asinh(x) behaves like the log
- for small x the function is close to linear in x

Transformation Example

```
library(patchwork)
v1 = seq(0, 1, length.out = 100)
v3 = seq(30, 3000, length = 100)
```



One Dimensional Clustering

```
# one dimensional k-means
kf = kmeansFilter("CD3all" = c("Pop1","Pop2"), filterId="myKmFilter")
fres = flowCore::filter(fcsBT, kf)
summary(fres)
```

```
## Pop1: 33429 of 91392 events (36.58%)
## Pop2: 57963 of 91392 events (63.42%)
```

Two dims. plot by the CD3 and CD56

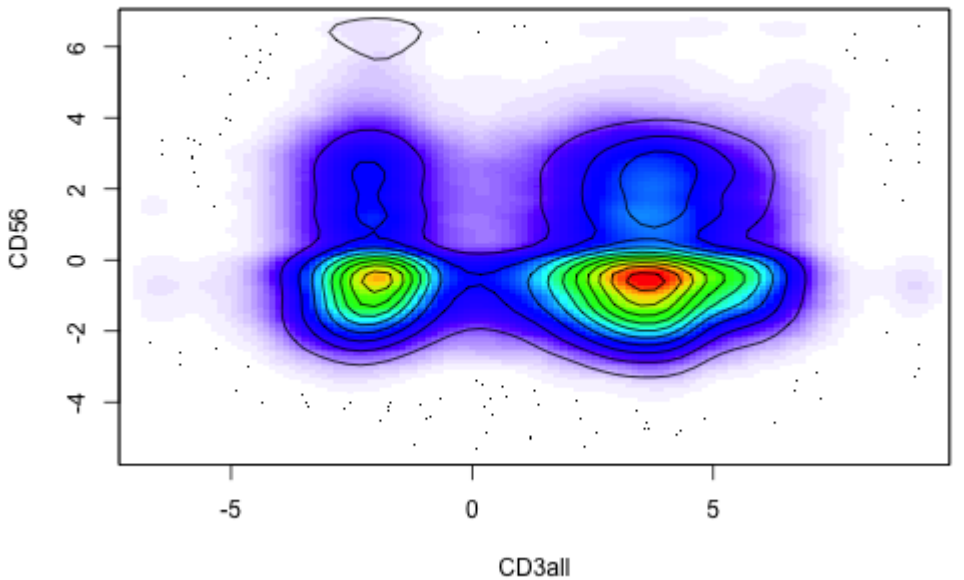
```
library("flowPeaks")
fp = flowPeaks(Biobase::exprs(fcsBT)[, c("CD3all", "CD56")])
```

```
##          step 0, set the intial seeds, tot.wss=17597.8
##          step 1, do the rough EM, tot.wss=11900.7 at 0.308807 sec
##          step 2, do the fine transfer of Hartigan-Wong Algorithm
##                   tot.wss=11846.3 at 0.576646 sec
```

```
plot(fp)
```

Use Contours and Shadding to Avoid Over Plotting

```
flowPlot(fcsBT, plotParameters = c("CD3all", "CD56"), logy = FALSE)
contour(fcsBT[, c(40, 19)], add = TRUE)
```



Density Based Clustering

- looks for regions of high density separated by sparser regions
- advantage of being able to cope with clusters that are not necessarily convex

```
library("dbscan")
mc5 = Biobase::exprs(fcsBT)[, c(15,16,19,40,33)]
head(mc5, 3)
```

```
##              CD4       CD8       CD20    CD3all      CD56
## [1,] -0.9547135 -0.278349 -1.2781241  4.002830  2.144493
## [2,] -0.5113526  1.110159 -0.1752901 -2.318107  1.272534
## [3,] -0.5421367  2.343449 -0.7316469  3.688635 -1.207639
```

```
res5 = dbscan::dbscan(mc5, eps = 0.65, minPts = 30)
mc5df = data.frame(mc5, cluster = as.factor(res5$cluster))
# how many cells assigned to clluster
table(mc5df$cluster)
```

```
##
##     0     1     2     3     4     5     6     7
## 77655   230  5114  4616  3310   207   231    29
```

Overlaping of contours highlights multidimensional nature of the clustering